Active Learning Through Chest X-ray Imaging

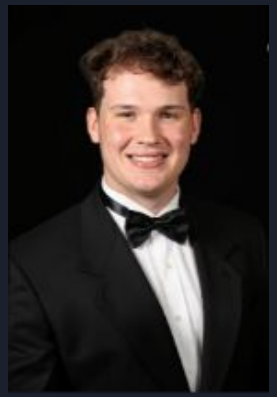
CS 4850-01/03/04, Fall 2025
Presented November 4th, 2025
Sharon Perry

Prototype Presentation Presented by

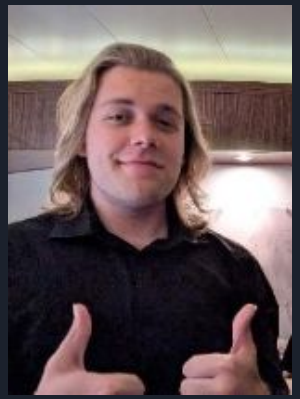
Josh Smith
Elijah Merrill
Matthew Hall
Noah Lane

Introductions

Josh Smith
Team Leader
Development



Elijah Merrill
Development



Noah Lane
Documentation

Mathew Hall
Documentation

Overview

- General Motives and Goals
- Required documents
- Code explanation
- Future works

Motive

- Originally idealed on Satellite Image Dataset
- The strive to have an outstanding project to present to the tech industry
- To experiment with new software and obtain an understanding
- To build an image classification system that would be dynamic and would work with many datasets rather than one

SDLC - Requirements

Functional Requirements

- Load and preprocess Chest X-ray dataset.
- Train baseline model using transfer learning.
- Implement active learning loop with uncertainty sampling.
- Evaluate performance (Accuracy, F1, AUC-ROC, AUC-PR).
- Provide user interface for training, labeling, and visualization.

Non-Functional Requirements

- Modular, maintainable, and well-documented codebase.
- Simple, intuitive user interface.
- Efficient performance on GPU-enabled systems.
- Cross-platform (Windows, macOS, Linux).
- Compatible with TensorFlow or PyTorch frameworks.

SDLC - Design

Design Considerations

- Uses public, locally stored datasets for privacy and performance.
- Requires modern OS and CPU for smooth image processing.
- Interface supports both experts and beginners.
- Future updates may enhance accuracy and efficiency.

Architectural Strategies

- Modular structure: Data, Training, Active Learning, Evaluation, UI.
- Python-based (PyTorch, NumPy, Pandas, Matplotlib).
- Employs CNNs and uncertainty sampling for model training.
- Focus on scalability, maintainability, and robust error handling.

SDLC - Development

Data Pipeline

- Indexer.py - builds table linking images
- Splitter.py - divides dataset by unique patient IDs
- Preprocess.py - applies resize / normalize / augmentation
- Dataset.py - loads transformed image-label pairs into PyTorch

Supporting Components

- Config.yaml - central parameters (paths, classes, image size, splits)
- Smoke.py - verifies entire pipeline functionality before training
- Frameworks/Libraries - Pytorch, Pandas, NumPy, Pillow, TorchVision, TQDM
- Dataset Source - NIH Chest X-ray (Kaggle)

Indexer

- Links each image filename to its full file path on disk
- Rename columns & standardizes labels so downstream code can use them cleanly
- Builds a DataFrame that maps various parameters
- Acts as a “single source of truth “ for dataset structure

```
#join the root path with CSV filename to get full path
csv_path = os.path.join(data_root, 'Data_Entry_2017.csv')
#check if the CSV file exists. if not, programs stops and raise error
if not os.path.exists(csv_path):
    raise FileNotFoundError(f"Missing {csv_path}")

#load the CSV metadata into a DataFrame
meta = pd.read_csv(csv_path)

#rename columns for cleaner access within python
meta = meta.rename(columns={
    "Image Index": "image_id", #filename of x-ray image
    "Finding Labels": "label_raw", #original string of disease labels
    "Patient ID": "patient_id", #id of the patient
    "View Position": "view" #x-ray view type (PA, AP, etc)
})
```

```
#walk through the directory and build a map of filenames -> absolute paths
path_map = {}
for root, _, files in os.walk(img_dir):
    for f in files:
        #include only png/jpp/jpeg files (ignoring others)
        if f.lower().endswith((".png", ".jpg", ".jpeg")):
            path_map[f] = os.path.join(root, f)

#map each image_id from the CSV to its actual file path
meta["path"] = meta["image_id"].map(path_map.get)
#create a boolean column showing which images are missing
meta["missing"] = meta["path"].isna()
```

Splitter

- Divides the dataset into three sets: train / validation / test sets
- Splits by unique Patient IDs rather than by individual images
- Ensures that no patient appears in multiple splits
- Adds a new “split” column to the dataset

```python
#split dataset into train/validation/test sets using patient ids rather than individual images to avoid data leaking where a patient
def split_by_patient(df: pd.DataFrame, ratios=(0.7,0.15,0.15), seed=42):  4 usages
    #safety check to ensure ratios sum to 1.0 (with tolerance)
    assert abs(sum(ratios) - 1.0) < 1e-6
    #create a reproducible random generator
    rng = np.random.default_rng(seed)
    #collect unique patient ids
    patients = df["patient_id"].dropna().unique()
    #shuffle patient ids randomly
    rng.shuffle(patients)

    #compute how many patients go into each split
    n = len(patients)
    n_train = int(ratios[0] * n)
    n_val = int(ratios[1] * n)

    #slice the shuffled list into three sets
    train_p = set(patients[:n_train])
    val_p = set(patients[n_train:n_train+n_val])
    test_p = set(patients[n_train+n_val:])
```

```python
    #helper function that assigns a split label based on patient id
    def assign(pid):
        if pid in train_p: return "train"
        if pid in val_p: return "val"
        return "test"

    #make a copy so the orginal DataFrame not modified directly
    df = df.copy()
    #apply the split assignment to every row
    df["split"] = df["patient_id"].map(assign)
    #sanity check to ensure we have exactly three split categories
    assert set(df["split"].unique()) == {"train", "val", "test"}
    #return updated DataFrame and a quick summary of patient counts
    return df, {"train": len(train_p), "val": len(val_p), "test": len(test_p)}
```

Preprocess

- Prepares image data for PyTorch
  - Uses torchvision.transforms()
  - Resizing
  - Cropping
  - Normalization
- Applies random flips and rotations to boost dataset diversity

```
#flips image horizontally and rotates image slightly
t += [transforms.RandomHorizontalFlip(p=0.5),
      transforms.RandomRotation(degrees=5)]
#scale the image to input size model expects
t += [transforms.Resize(input_size, antialias=True),
      transforms.ToTensor()]
#add normalization (optional)
if use_imagenet_norm:
    t += [transforms.Normalize(mean=[0.485, 0.456, 0.406],
                               std=[0.229, 0.224, 0.225])]
#combine all transformations into a single callable pipeline
return transforms.Compose(t)
```

Dataset

- Connects preprocessed data to the model's training process
- Loads x-ray images and converts them to RGB
- Turns images and labels into PyTorch tensors

```
class CXRDataset(Dataset):
    def __init__(self, index_df: pd.DataFrame, split: str, class_names, transforms):
        self.df = index_df[index_df["split"] == split].reset_index(drop=True)
        self.class_names = class_names
        self.t = transforms

    def __len__(self): return len(self.df)

    def __getitem__(self, i: int):
        row = self.df.iloc[i]
        img = Image.open(row.path).convert("RGB") #ensure 3-channel
        x = self.t(img)
        y = torch.tensor(row[self.class_names].values.astype("float32"))
        meta = {"image_id": row.image_id, "patient_id": row.patient_id}
        return x, y, meta
```

Config

- .yaml file that controls all main settings – dataset paths, preprocessing, and data splits

- Allows for pipeline easy configuration

```yaml
data:
  # list of class names from the dataset
  class_names:
    - Atelectasis
    - Cardiomegaly
    - Effusion
    - Infiltration
    - Mass
    - Nodule
    - Pneumonia
    - Pneumothorax
    - Consolidation
    - Edema
    - Emphysema
    - Fibrosis
    - Pleural_Thickening
    - Hernia
```

Smoke

- End-to-end integration test of the entire data pipeline
- Confirms tensor dimensions (N, C, H, W) match the expected CNN input size
- Runs before any real model training to catch issues early

```python
from torch.utils.data import DataLoader
from src.data.indexer import build_index, filter_views
from src.data.splitter import split_by_patient, save_splits_json
from src.data.preprocess import make_transforms
from src.data.dataset import CXRDataset

def main(cfg_path = "config.yaml"):  1 usage
    cfg = yaml.safe_load(open(cfg_path))
    root = cfg["paths"]["data_root"]
    cls  = cfg["data"]["class_names"]
    keep = cfg["data"]["keep_views"]
    h, w = cfg["model"]["input_size"]
```

Outputs

```
(.venv312) PS D:\School Files\School Notes\Fall 2025\CS 4850\Project Testing On My System\active_learning_system_on_host> python -m src.data.smoke
Building index:
Images in CSV: 112,120; missing paths: 0
Splitting by patient:
Patients per split: {'train': 21563, 'val': 4620, 'test': 4622}
D:\School Files\School Notes\Fall 2025\CS 4850\Project Testing On My System\active_learning_system_on_host\.venv312\Lib\site-packages\torch\utils\data\dataloader.py:668: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
  warnings.warn(warn_msg)
Train batch: x=(8, 3, 224, 224), y=(8, 14) (N,C,H,W) = torch.Size([8, 3, 224, 224])
Example metas: {'image_id': ['00020691_003.png', '00002350_010.png', '00019407_007.png', '00015071_002.png', '00006013_003.png', '00019718_000.png', '00012280_013.png', '00021031_001.png'], 'patient_id': tensor([20691,  2350, 19407, 15071,  6013, 19718, 12280, 21031])}
(.venv312) PS D:\School Files\School Notes\Fall 2025\CS 4850\Project Testing On My System\active_learning_system_on_host>
```

Outputs

splits.json

File Edit View


```
{
  "counts": {
    "train": 78934,
    "val": 16871,
    "test": 16315
  },
  "patients_per_split": {
    "test": 4622,
    "train": 21563,
    "val": 4620
  }
}
```


image_id - str	label_raw - str	Follow-up # - i64	patient_id - i64	Patient Age - i64	Patient Gender - str	view - str	OriginalImage[Width - i64	Height] - i64	OriginalImageI
00000001_000.png	Cardiomegaly	0	1	58	M	PA	2682	2749	0.143
00000001_001.png	Cardiomegaly|Emphysema	1	1	58	M	PA	2894	2729	0.143
00000001_002.png	Cardiomegaly|Effusion	2	1	58	M	PA	2500	2048	0.168
00000002_000.png	No Finding	0	2	81	M	PA	2500	2048	0.171
00000003_000.png	Hernia	0	3	81	F	PA	2582	2991	0.143
00000003_001.png	Hernia	1	3	74	F	PA	2500	2048	0.168
00000003_002.png	Hernia	2	3	75	F	PA	2048	2500	0.168
00000003_003.png	Hernia|Infiltration	3	3	76	F	PA	2698	2991	0.143
00000003_004.png	Hernia	4	3	77	F	PA	2500	2048	0.168
00000003_005.png	Hernia	5	3	78	F	PA	2686	2991	0.143
00000003_006.png	Hernia	6	3	79	F	PA	2992	2991	0.143
00000003_007.png	Hernia	7	3	80	F	PA	2582	2905	0.143
00000004_000.png	Mass|Nodule	0	4	82	M	AP	2500	2048	0.168
00000005_000.png	No Finding	0	5	69	F	PA	2048	2500	0.168
00000005_001.png	No Finding	1	5	69	F	AP	2500	2048	0.168
00000005_002.png	No Finding	2	5	69	F	AP	2500	2048	0.168
00000005_003.png	No Finding	3	5	69	F	PA	2992	2991	0.143
00000005_004.png	No Finding	4	5	70	F	PA	2986	2991	0.143
00000005_005.png	No Finding	5	5	70	F	PA	2514	2991	0.143
00000005_006.png	Infiltration	6	5	70	F	PA	2992	2991	0.143
00000005_007.png	Effusion|Infiltration	7	5	70	F	PA	2566	2681	0.143
00000006_000.png	No Finding	0	6	81	M	PA	2500	2048	0.168
00000007_000.png	No Finding	0	7	82	M	PA	2500	2048	0.168
00000008_000.png	Cardiomegaly	0	8	69	F	PA	2048	2500	0.171
00000008_001.png	No Finding	1	8	70	F	PA	2048	2500	0.171
00000008_002.png	Nodule	2	8	73	F	PA	2048	2500	0.168
00000009_000.png	Emphysema	0	9	73	M	PA	2992	2991	0.143
00000010_000.png	Infiltration	0	10	84	F	PA	2992	2991	0.143
00000011_000.png	Effusion	0	11	75	M	PA	2638	2449	0.143
00000011_001.png	No Finding	1	11	75	M	PA	2500	2048	0.168
00000011_002.png	No Finding	2	11	75	M	PA	2714	2781	0.143
00000011_003.png	No Finding	3	11	75	M	PA	2500	2048	0.168
00000011_004.png	No Finding	4	11	75	M	PA	2500	2048	0.168

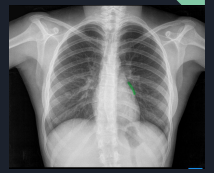
Set-Up Process

- The project includes six essential source files found in the “source code” folder of the GitHub repository.
- The dataset can be downloaded from Kaggle – NIH Chest X-Rays. After extraction, store the “archive” folder alongside the source files or record its full path for configuration.
- The system requires Python version 3.10 or later and several key packages: torch, torchvision, torchaudio, pandas, pyarrow, numpy, tqdm, and matplotlib.
- Verify that dataset and output paths match your local directory structure, updating any file references as needed.
- To confirm successful setup, run python smoke.py.

Database Connection

- The system does not use a traditional database connection. Instead, it relies on a file-based data storage approach managed locally on the user’s system.
- A publicly available chest X-ray dataset serves as the primary data source for model training, validation, and testing.
- All image files and labels are loaded directly from local directories or structured folders rather than through a database.
- This design ensures simpler configuration, faster access, and greater portability across systems.

Future Development Overview (CNN)



Testing
VS
Training

A	B	C	D	E
37	21	28		45
20	24	21	1	20
17	13	18		27
5	33	17		18
14	16	12	2	28
14	13	14	1	26
16	13	19	2	18
12	16	9	1	28
11	15	12		12
5	7	12		15

Input dataset and/or image

Pre-processing and data labeling

Transfer data into usable data

Node (1, 1)

Node (n, 1)

Node (1, m)

Node (n, m)

Confidence value received from CNN

Interpre-tation of the data

"Healthy"

Training and Testing Data

- 70% training data - data to train the CNN
- 15% validation data - data to sure that the data fits in accordance to the rest of the set
- 15% testing data - data put against the training data to see if our program is working correctly

Summary

- What motivated us
- SDLC Documents
- Data processing pipeline
- Future work/phases
- We submitted our powerpoint slides prior to this presentation.